Flex 3 Cookbook™

Joshua Noble and Todd Anderson

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Adobe Developer Library, a copublishing partnership between O'Reilly Media Inc., and Adobe Systems, Inc., is the authoritative resource for developers using Adobe technologies. These comprehensive resources offer learning solutions to help developers create cutting-edge interactive web applications that can reach virtually anyone on any platform.

With top-quality books and innovative online resources covering the latest tools for rich-Internet application development, the *Adobe Developer Library* delivers expert training straight from the source. Topics include ActionScript, Adobe Flex®, Adobe Flash®, and Adobe Acrobat®.

Get the latest news about books, online resources, and more at *http://adobedeveloperlibrary.com*.

This excerpt is protected by copyright law. It is your responsibility to obtain permissions necessary for any proposed use of this material. Please direct your inquiries to permissions@oreilly.com.

Table of Contents

Preface ..... **xvii**

1. Flex and ActionScript Basics **1**
1.1 Create a Flex Project in Flex Builder 2
1.2 Create a Flex Library Project in Flex Builder 7
1.3 Create an ActionScript Project 9
1.4 Set Compiler Options for the MXML Compiler in Flex Builder 11
1.5 Compile a Flex Project Outside of Flex Builder 14
1.6 Add an Event Listener in MXML 16
1.7 Set Properties of a Child Defined in MXML in ActionScript 18
1.8 Define Arrays and Objects 20
1.9 Set the Scope of Variables in ActionScript 21
1.10 Create a Component in ActionScript 24
1.11 Use Event Bubbling 26
1.12 Use a Code-Behind Model to Separate MXML and ActionScript 28
1.13 Make Properties of a Component Bindable 29
1.14 Use Custom Events and Dispatch Data with Events 30
1.15 Listen for a Keyboard Event 32
1.16 Define Optional Parameters for Methods 33
1.17 Determine the Type of an Object 34
1.18 Define and Implement an Interface 35

2. Menus and Controls **39**
2.1 Listen to a Button Click 39
2.2 Create a Set of Buttons That Toggle 42
2.3 Use a ColorPicker to Set Canvas Color 45
2.4 Load a SWF by Using the SWFLoader 46
2.5 Set Tab Indexes for Components 46
2.6 Set a labelFunction for a Control 47
2.7 Provide Data for Menus 48
2.8 Dynamically Populate Menus 50
2.9 Create EventHandlers for Menu-Based Controls 52

2.10 Display an Alert in an Application 53
2.11 Use the Date from a Calendar Control 55
2.12 Display and Position Multiple Pop-ups 56
2.13 Create a Custom Border for a Pop-up Window 59
2.14 Handle focusIn and focusOut Events 60

3. Containers 63
3.1 Position Children by Using Layout Management 63
3.2 Position and Size Containers via Percentage Positioning 65
3.3 Track Mouse Position Within Different Coordinate Systems 66
3.4 Dynamically Add and Remove Children from a Container 67
3.5 Use Constraint-Based Layout for Containers 69
3.6 Set Maximum and Minimum Sizes for Children Within Containers 70
3.7 Specify Constraint Rows and Columns for a Container 71
3.8 Create Layout Flows for Text Using Constraints 73
3.9 Control Scrolling and Overflow Within Containers 75
3.10 Control the Layout of Box Components 77
3.11 Use Containers for Initialization 78
3.12 Create a TitleWindow 79
3.13 Control a ViewStack via a LinkBar 80
3.14 Bind the Selected Index of a ViewStack to a Variable 81
3.15 Use Delayed Instantiation to Improve Startup Time 83
3.16 Create and Control Resizable Containers 84
3.17 Create, Enable, and Disable TabControls Within a TabNavigator 85
3.18 Create a TabNavigator with Closeable Tabs 87
3.19 Create and Control an Alert 88
3.20 Size and Position a Dialog Box Based on Its Calling Component 90
3.21 Manage Multiple Pop-up Dialog Boxes 91
3.22 Scroll to a Specific Child in a Container 93
3.23 Create a Template Using IDeferredInstance 94
3.24 Manually Lay Out a Container 97
3.25 Measure and Alter Container Size 101
3.26 Control the Visibility and Layout of Children 102
3.27 Create a Tile Container with Simple Reorganization 104
3.28 Set a Background Image and Rounded Corners in an HBox 106
3.29 Control Positioning and Scrolling of Child Components 107

4. Text 111
4.1 Correctly Set the Value of a Text Object 111
4.2 Bind a Value to TextInput 113
4.3 Create a Suggestive TextInput 114

4.4 Create an In-Place Editor 115
4.5 Determine All Fonts Installed on a User's Computer 116
4.6 Create a Custom TextInput 118
4.7 Set the Style Properties for Text Ranges 119
4.8 Display Images and SWFs in HTML 120
4.9 Highlight User-Input Text in a Search Field 121
4.10 Manipulate Characters as Individual Graphics 122
4.11 Specify Styles for HTML in a TextField 126
4.12 Use the RichTextEditor 127
4.13 Apply Embedded Fonts with HTML 128
4.14 Add a Drop Shadow to Text in a Text Component 129
4.15 Find the Last Displayed Character in a TextArea 131

5. Lists, Tiles, and Trees 133
5.1 Create an Editable List 133
5.2 Set Icons for Items in a List 135
5.3 Add Effects to a List to Indicate Changes 136
5.4 Set a Basic Item Renderer for a TileList 138
5.5 Set XML Data for a Tree 140
5.6 Create an Item Renderer for a Tree 141
5.7 Use Complex Data Objects in a Tree Control 143
5.8 Allow Only Certain Items in a List to Be Selectable 148
5.9 Format and Validate Data Added in a List's Item Editor 151
5.10 Track All Selected Children in a TileList 153
5.11 Use and Display Null Items in an Item Renderer 156
5.12 Create a Right-Click Menu for a List 157
5.13 Customize the Appearance of a Selection in a List 159

6. DataGrid and Advanced DataGrid 161
6.1 Create Custom Columns for a DataGrid 161
6.2 Specify Sort Functions for DataGrid Columns 164
6.3 Enable Multicolumn Sorting in a DataGrid 166
6.4 Filter Items in a DataGrid 168
6.5 Create Custom Headers for an AdvancedDataGrid 170
6.6 Handle Events from a DataGrid/AdvancedDataGrid 173
6.7 Select Items in an AdvancedDataGrid 176
6.8 Enable Drag-and-Drop in a DataGrid 179
6.9 Edit Items in a DataGrid 180
6.10 Search Within a DataGrid and Autoscroll to the Match 182
6.11 Generate a Summary for Flat Data by Using GroupingCollection 184
6.12 Create an Async Refresh for a GroupingCollection 187

7. Renderers and Editors 193
7.1 Create Your Own Renderers 193
7.2 Use the ClassFactory to Generate Renderers 196
7.3 Access the Component That Owns a Renderer 200
7.4 Create a Single Component to Act as Renderer and Editor 203
7.5 Create an Item Editor to Handle Data with Multiple Fields 205
7.6 Display SWF Objects as Items in a Menu by Using an Item Renderer 207
7.7 Select a DataGrid Column with a CheckBox Header Renderer 209
7.8 Create a Self-Contained CheckBox itemRenderer for Use in a DataGrid 212
7.9 Efficiently Set Images in a Renderer 214
7.10 Use Runtime Styling with itemRenderers and itemEditors 217
7.11 Use States and Transitions with an itemEditor 219
7.12 Create a CheckBox Tree Control 221
7.13 Resize Renderers Within a List 226

8. Images, Bitmaps, Videos, Sounds 229
8.1 Load and Display an Image 230
8.2 Create a Video Display 231
8.3 Play and Pause an MP3 File 232
8.4 Create a Seek Bar for a Sound File 234
8.5 Blend Two Images 235
8.6 Apply a Convolution Filter to an Image 238
8.7 Send Video to an FMS Instance via a Camera 240
8.8 Access a User's Microphone and Create a Sound Display 242
8.9 Smooth Video Displayed in a Flex Application 244
8.10 Check Pixel-Level Collisions 245
8.11 Read and Save a User's Webcam Image 248
8.12 Use Blend Modes with Multiple Images 250
8.13 Handle Cue Points in FLV Data 251
8.14 Create a Video Scrubber 253
8.15 Read ID3 Data from an MP3 File 255
8.16 Display a Custom Loader while Loading Images 257
8.17 Enable Image Upload in Flex 258
8.18 Compare Two Bitmap Images 260

9. Skinning and Styling 263
9.1 Use CSS to Style Components 264
9.2 Override the Default Application Style 266
9.3 Embed Styles by Using CSS 268
9.4 Override Base Style Properties 269
9.5 Customize Styles at Runtime 270

9.6 Load CSS at Runtime 272
9.7 Declare Styles at Runtime 274
9.8 Create Custom Style Properties for Components 276
9.9 Use Multiple Themes in the Same Application 279
9.10 Compile a Theme SWC 280
9.11 Use Embedded Fonts 283
9.12 Embed Fonts from a SWF File 285
9.13 Skin with Embedded Images 289
9.14 Apply Skins from a SWF File 291
9.15 Programmatically Skin a Component 295
9.16 Programmatically Skin a Stateful Control 299
9.17 Create Animated Skins from a SWF File 302
9.18 Customize the Preloader 306

10. Dragging and Dropping 313
10.1 Use the DragManager Class 313
10.2 Specify a Drag Proxy 317
10.3 Drag and Drop Within a List 319
10.4 Drag and Drop Between Lists 322
10.5 Enable and Disable Drag Operations 323
10.6 Customize the DragImage of a List-Based Control 326
10.7 Customize the Drop Indicator of a List-Based Control 329

11. States 333
11.1 Set Styles and Properties in a State 334
11.2 Create Transitions to Enter and Leave States 335
11.3 Use the AddChildAction and RemoveChildAction 337
11.4 Filter Transitions to Affect Only Certain Types of Children 339
11.5 Apply Parts of a Transition to Certain Children 341
11.6 Base a State on Another State 344
11.7 Integrate View States with HistoryManagement 345
11.8 Use Deferred Instance Factories with States 347
11.9 Use Data Binding with Objects Added in a State 349
11.10 Add and Remove Event Listeners in State Changes 351
11.11 Add View States to a Flash Component 352
11.12 Work with State Change Events 355
11.13 Dynamically Generate and Use New States and Transitions 357
11.14 Create Custom Actions to Use in a State 358

12. Effects 361
12.1 Call an Effect in MXML and in ActionScript 362
12.2 Build a Custom Effect 363
12.3 Create Parallel Series or Sequences of Effects 365

12.4 Pause, Reverse, and Restart an Effect 366
12.5 Create Custom Effect Triggers 367
12.6 Create Tween Effects 368
12.7 Use the DisplacementMapFilter Filter in a Flex Effect 371
12.8 Create an AnimateColor Effect 375
12.9 Use the Convolution Filter to Create a Tween 376

13. Collections 381
13.1 Add, Sort, and Retrieve Data from an ArrayCollection 381
13.2 Filter an ArrayCollection 384
13.3 Determine When an Item Is Modified in an ArrayCollection 385
13.4 Create a GroupingCollection 386
13.5 Create a Hierarchical Data Provider for a Control 387
13.6 Navigate a Collection Object and Save Your Position 392
13.7 Create a HierarchicalViewCollection Object 394
13.8 Filter and Sort an XMLListCollection 396
13.9 Sort on Multiple Fields in a Collection 398
13.10 Sort on Dates in a Collection 399
13.11 Create a Deep Copy of an ArrayCollection 400
13.12 Use Data Objects with Unique IDs 402

14. Data Binding 405
14.1 Bind to a Property 406
14.2 Bind to a Function 408
14.3 Create a Bidirectional Binding 411
14.4 Bind to Properties by Using ActionScript 411
14.5 Use Bindable Property Chains 415
14.6 Bind to Properties on XML by Using E4X 417
14.7 Create Customized Bindable Properties 419
14.8 Bind to a Generic Object 423
14.9 Bind to Properties on a Dynamic Class 425

15. Validation, Formatting, and Regular Expressions 431
15.1 Use Validators and Formatters with TextInput and TextArea Controls 432
15.2 Create a Custom Formatter 434
15.3 Create a More-International Zip Code Validator by Using Regular Expressions 436
15.4 Create a Validator to Validate UPCs 438
15.5 Validate Combo Boxes and Groups of Radio Buttons 440
15.6 Show Validation Errors by Using ToolTips in a Form 443
15.7 Use Regular Expressions for Locating Email Addresses 446
15.8 Use Regular Expressions for Matching Credit Card Numbers 446

15.9 Use Regular Expressions for Validating ISBNs 447
15.10 Create Regular Expressions by Using Explicit Character Classes 447
15.11 Use Character Types in Regular Expressions 448
15.12 Match Valid IP Addresses by Using Subexpressions 450
15.13 Use Regular Expressions for Different Types of Matches 451
15.14 Match Ends or Beginnings of Lines with Regular Expressions 453
15.15 Use Back-References 453
15.16 Use a Look-Ahead or Look-Behind 455

16. Working with Services and Server-Side Communication 457
16.1 Configure an HTTPService 458
16.2 Use RESTful Communication Between Flex Applications 460
16.3 Configure and Connect to a RemoteObject 461
16.4 Use Flex Remoting with AMFPHP 1.9 464
16.5 Use the IExternalizable Interface for Custom Serialization 468
16.6 Track Results from Multiple Simultaneous Service Calls 470
16.7 Use Publish/Subscribe Messaging 471
16.8 Register a Server-Side Data Type Within a Flex Application 472
16.9 Communicate with a WebService 474
16.10 Add a SOAP Header to a Request to a WebService 476
16.11 Parse a SOAP Response from a WebService 477
16.12 Communicate Securely with AMF by Using SecureAMFChannel 479
16.13 Send and Receive Binary Data via a Binary Socket 480
16.14 Communicate Using an XMLSocket 482

17. Browser Communication 483
17.1 Link to an External URL 483
17.2 Work with FlashVars 484
17.3 Invoke JavaScript Functions from Flex 486
17.4 Invoke ActionScript Functions from JavaScript 487
17.5 Change the HTML Page Title via BrowserManager 489
17.6 Parse the URL via BrowserManager 490
17.7 Deep-Link to Data via BrowserManager 492
17.8 Deep-Link Containers via BrowserManager 493
17.9 Implement Custom History Management 496

18. Modules and Runtime Shared Libraries 499
18.1 Create a Runtime Shared Library 500
18.2 Use Cross-Domain Runtime Shared Libraries 503
18.3 Use the Flex Framework as a Runtime Shared Library 505
18.4 Optimize a Runtime Shared Library 507
18.5 Create an MXML-Based Module 508

18.6 Create an ActionScript-Based Module 510
18.7 Load a Module by Using ModuleLoader 512
18.8 Use ModuleManager to Load Modules 514
18.9 Load Modules from Different Servers 516
18.10 Communicate with a Module 518
18.11 Pass Data to Modules by Using Query Strings 523
18.12 Optimize Modules by Using Linker Reports 526

19. The Adobe Integrated Runtime API 529
19.1 Create an AIR Application Leveraging the Flex Framework 530
19.2 Understand the AIR Command-Line Tools 532
19.3 Open and Manage Native Windows 537
19.4 Create Native Menus 541
19.5 Read and Write to a File 544
19.6 Serialize Objects 546
19.7 Use the Encrypted Local Store 551
19.8 Browse for Files 553
19.9 Use the AIR File System Controls 555
19.10 Use the Native Drag-and-Drop API 558
19.11 Interact with the Operating System Clipboard 562
19.12 Add HTML Content 564
19.13 Cross-Script Between ActionScript and JavaScript 567
19.14 Work with Local SQL Databases 570
19.15 Detect and Monitor a Network Connection 575
19.16 Detect User Presence 577
19.17 Create System Tray and Dock Applications 578

20. Unit Testing with FlexUnit 581
20.1 Create an Application That Uses the FlexUnit Framework 582
20.2 Create an Application to Run FlexUnit Tests 582
20.3 Create a FlexUnit Test Case 584
20.4 Add a Test Case to a Test Suite 587
20.5 Run Code Before and After Every Test 588
20.6 Share Test Data Between Test Cases 590
20.7 Handle Events in a Test Case 592
20.8 Test Visual Components with FlexUnit 595
20.9 Install and Configure Antennae 605
20.10 Generate Automated Test Suites 607

21. Compiling and Debugging 611
21.1 Use Trace Statements Without Flex Builder 611
21.2 Use the Component Compiler 612
21.3 Install the Flex Ant Tasks 614

21.4 Use the compc and mxmlc Tasks in the Flex Ant Tasks 615
21.5 Compile and Deploy Flex Applications That Use RSLs 615
21.6 Create and Monitor Expressions in Flex Builder Debugging 618
21.7 Install the Ant View in the Stand-Alone Version of Flex Builder 620
21.8 Create an Ant Build File for Automating Common Tasks 621
21.9 Compile a Flex Application by Using mxmlc and Ant 622
21.10 Generate Documentation by Using ASDoc and Ant 624
21.11 Compile Flex Applications by Using Rake 625
21.12 Use ExpressInstall for Your Application 626
21.13 Use Memory Profiling with Flex Builder 3 to View Memory Snapshots 628

22. Configuration, Internationalization, and Printing 631
22.1 Add an International Character Set to an Application 631
22.2 Use a Resource Bundle to Localize an Application 633
22.3 Use the ResourceManager for Localization 637
22.4 Use Resource Modules for Localization 639
22.5 Support IME Devices 642
22.6 Detect a Screen Reader 644
22.7 Create a Tabbing Reading Order for Accessibility 645
22.8 Print Selected Items in an Application 646
22.9 Format Application Content for Printing 647
22.10 Control Printing of Unknown Length Content over Multiple Pages 649
22.11 Add a Header and a Footer When Printing 650

Index 655

CHAPTER 8
Images, Bitmaps, Videos, Sounds

Images, bitmaps, videos, and sounds is a mouthful and a far wider range of topics than could be adequately covered in a single chapter, so this one concentrates on answering the most common questions. As Flash becomes the primary method of delivering video over the Internet and the use of the Flex Framework in creating photo and MP3 applications increases, understanding how to work with all of these elements becomes more and more important.

The Flash Player offers multiple levels of tools for dealing with images and sound. The first avenue of control contains the `Image` and `VideoDisplay` classes, MXML classes that simplify much of dealing with images and video and enable you to quickly integrate these assets into your application. The next step down is the `flash.media` package, which houses the `Video`, `Sound`, `SoundTransform`, `Camera`, and `Microphone` classes; their corollaries, `Loader`, `NetConnection`, and `NetStream`, are in the `flash.net` package. These classes provide much finer control over the integration of sound, video, and images into an application and require slightly more time to perfect. Finally, you can reach down to the bytes that make up all data in the Flash Player: the `BitmapData` classes and the `ByteArray` classes. These enable you not only to manipulate the bitmap data of the images that you load into the Flash Player, but also to create new bitmaps and stream the data out.

Many of the examples in this chapter manipulate images and videos as bitmap data. This is not nearly as difficult as it sounds, because the Flash Player provides numerous convenience methods for working with the `BitmapData` class, and manipulating the bitmap data directly greatly increases the efficiency of your application. You'll also be working extensively with the `NetStream` class, for handling video and users' microphones and cameras. `NetStream` is an effective way of streaming information both to and from server-side applications.

8.1 Load and Display an Image

Problem

You need to display an image in a Flex component.

Solution

Use either an `Embed` statement to compile the image into the SWF file or load the image at runtime.

Discussion

Flex supports importing GIF, JPEG, PNG, and SWF files at runtime or at compile time, and SVG files at compile time through embedding. The method you choose depends on the file types of your images and your application parameters. Any embedded images are already part of the SWF file and so don't require any time to load. The trade-off is the size that they add to your application, which slows the application initialization process. Extensive use of embedded images also requires you to recompile your applications whenever your image files change.

Alternatively, you can load the resource at runtime by either setting the `source` property of an image to a URL or by using `URLRequest` objects and making the result of the load operation a `BitmapAsset` object. You can load a resource from the local file system in which the SWF file runs, or you can access a remote resource, typically through an HTTP request over a network. These images are independent of your application; you can change them without needing to recompile as long as the names of the modified images remain the same.

Any SWF file can access only one type of external resource, either local or over a network; it cannot access both types. You determine the type of access allowed by the SWF file by using the `use-network` flag when you compile your application. When the `use-network` flag is set to false, you can access resources in the local file system, but not over the network. The default value is true, which allows you to access resources over the network, but not in the local file system.

To embed an image file, use the `Embed` metadata property:

```
[Embed(source="../assets/flag.png")]
private var flag:Class;
```

Now this `Class` object can be set as the source for an image:

```
var asset:BitmapAsset = new flag() as BitmapAsset;
img3rd.source = asset;
```

Alternatively, you can set the property of the source to a local or external file system:

```
<mx:Image source="http://server.com/beach.jpg"/>
```

The full example follows:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;

            [Embed(source="../assets/flag.png")]
            private var flag:Class;

            private function imgMod():void
            {
                var asset:BitmapAsset = new flag() as BitmapAsset;
                img3rd.source = asset;
            }

        ]]>
    </mx:Script>
    <mx:Image source="../assets/flag.png"/>
    <mx:Image source="{flag}"/>
    <mx:Image id="img3rd" creationComplete="imgMod()"/>
</mx:VBox>
```

8.2 Create a Video Display

Problem

You need to display an FLV file in your application.

Solution

Use the `VideoDisplay` class in your application and use `Button` objects to play and pause the application if desired.

Discussion

The `VideoDisplay` class wraps a `flash.media.Video` object and simplifies adding video to that object considerably. The `source` attribute of the `VideoDisplay` is set to the URL of an FLV file, and the `autoplay` parameter is set to true so that when the `NetStream` has been properly instantiated and the video information begins streaming to the player, the video will begin playing:

```
<mx:VideoDisplay source="http://localhost:3001/Trailer.flv" id="vid" autoplay="true"/>
```

In the following example, buttons are set up to play, pause, and stop the video by using the methods defined by the `VideoDisplay` class:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:VideoDisplay source="http://localhost:3001/Trailer.flv" id="vid" autoPlay=
"false" autoRewind="true"/>
    <mx:HBox>
            <mx:Button label="Play" click="vid.play();"/>
```

```
            <mx:Button label="Pause" click="vid.pause();"/>
            <mx:Button label="Stop" click="vid.stop();"/>
        </mx:HBox>
</mx:VBox>
```

8.3 Play and Pause an MP3 File

Problem

You want to allow a user to play a series of MP3 files.

Solution

Use the `Sound` and `SoundChannel` classes and load new files by using progressive download when the user selects a new MP3 file.

Discussion

The `play` method of the `Sound` class returns a `SoundChannel` object that provides access to methods and properties that control the balance or right and left volume of the sound, as well as methods to pause and resume a particular sound.

For example, let's say your code loads and plays a sound file like this:

```
var snd:Sound = new Sound(new URLRequest("sound.mp3"));
var channel:SoundChannel = snd.play();
```

You cannot literally pause a sound during playback in ActionScript; you can only stop it by using the `SoundChannel stop` method. You can, however, play a sound starting from any point. You can record the position of the sound at the time it was stopped, and then replay the sound starting at that position later.

While the sound plays, the `SoundChannel.position` property indicates the point in the sound file that's currently being played. Store the `position` value before stopping the sound from playing:

```
var pausePosition:int = channel.position;
channel.stop();
```

To resume the sound, pass the previously stored `position` value to restart the sound from the same point it stopped at before:

```
channel = snd.play(pausePosition);
```

The following complete code listing provides a combo box to allow the user to select different MP3 files, pause, and stop playback by using the `SoundChannel` class:

```
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;
```

```
            public var sound:Sound;
            public var chan:SoundChannel;
            public var pausePos:int = 0;

            private const server:String = "http://localhost:3001/"

            private var dp:ArrayCollection = new ArrayCollection(["Plans.mp3",
"Knife.mp3", "Marla.mp3", "On a Neck, On a Spit.mp3", "Colorado.mp3"])

            private function loadSound():void {
                if(chan != null) {
                    //make sure we stop the sound; otherwise, they'll overlap
                    chan.stop();
                }
                //re-create sound object, flushing the buffer, and readd the event
listener
                sound = new Sound();
                sound.addEventListener(Event.SOUND_COMPLETE, soundComplete);
                var req:URLRequest = new URLRequest(server + cb.selectedItem as
String);
                sound.load(req);
                pausePos = 0;
                chan = sound.play();
            }
            //
            private function soundComplete(event:Event):void {
                cb.selectedIndex++;
                sound.load(new URLRequest(server + cb.selectedItem as String));
                chan = sound.play();
            }

            private function playPauseHandler():void
            {
                if(pausePlayBtn.selected){
                    pausePos = chan.position;
                    chan.stop();
                } else {
                    chan = sound.play(pausePos);
                }
            }

        ]]>
    </mx:Script>
    <mx:ComboBox creationComplete="cb.dataProvider=dp" id="cb" change="loadSound()"/>
    <mx:Button label="start" id="pausePlayBtn" toggle="true" click=
"playPauseHandler()"/>
    <mx:Button label="stop" click="chan.stop()"/>
</mx:HBox>
```

8.4 Create a Seek Bar for a Sound File

Problem

You need to create a seek control for a user to seek different parts of an MP3 file and a volume control to change the volume of the MP3 playback.

Solution

Pass a `time` parameter to the `Sound play` method to begin playing the file from that point. This creates a new `SoundTransform` object that should be set as the `soundTransform` of the `SoundChannel`.

Discussion

The `play` method of the sound file accepts a start-point parameter:

```
public function play(startTime:Number = 0, loops:int = 0, sndTransform: SoundTransform
= null):SoundChannel
```

This creates a new `SoundChannel` object to play the sound returns that object, which you access to stop the sound and monitor the volume. (To control the volume, panning, and balance, access the `SoundTransform` object assigned to the `SoundChannel`.)

To control the volume of the sound, pass the `SoundTransform` object to the `SoundChannel`. We create a new `SoundTransform` object with the desired values and pass it to the `SoundChannel` that is currently playing:

```
var trans:SoundTransform = new SoundTransform(volumeSlider.value);
chan.soundTransform = trans;
```

The `SoundTransform` class accepts the following parameters:

```
SoundTransform(vol:Number = 1, panning:Number = 0)
```

The `panning` values range from –1.0, indicating a full pan left (no sound coming out of the right speaker) to 1.0, indicating a full pan right. A full code listing is shown here:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="loadSound()">
    <mx:Script>
        <![CDATA[

            private var sound:Sound;
            private var chan:SoundChannel;

            private function loadSound():void {
                sound = new Sound(new URLRequest("http://localhost:3001/Plans.mp3"));
                chan = sound.play();
            }

            private function scanPosition():void {
                chan.stop();
```

```
                //divide by 10 because the Slider values go from 0 - 10 and we want
a value
                //between 0 - 1.0
                chan = sound.play(positionSlider.value/10 * sound.length);
            }

            private function scanVolume():void
            {
                var trans:SoundTransform = new SoundTransform(volumeSlider.value,
(panSlider.value - 5)/10);
                chan.soundTransform = trans;
            }

        ]]>
    </mx:Script>
    <mx:Label text="Position"/>
    <mx:HSlider change="scanPosition()" id="positionSlider"/>
    <mx:Label text="Volume"/>
    <mx:HSlider change="scanVolume()" id="volumeSlider"/>
    <mx:Label text="Pan"/>
    <mx:HSlider change="scanVolume()" id="panSlider"/>
</mx:VBox>
```

8.5 Blend Two Images

Problem

You want to manipulate and combine multiple images at runtime and use filters to alter those images.

Solution

Cast the images as `BitmapData` objects and use the `combine` method of the `BitmapData` class to combine all the data in the two bitmaps into a new image.

Discussion

The `BitmapData` and `Bitmap` classes are powerful tools for manipulating images at runtime and creating new effects. The two classes are frequently used in tandem but are quite different. `BitmapData` encapsulates the actual data of the image, and `Bitmap` is a display object that can be added to the display list. The `BitmapData` object is created and drawn into as shown here:

```
var bitmapAsset:BitmapAsset = new BitmapAsset(img1.width, img1.height);
bitmapAsset.draw(img1);
```

First, set the height and width of the `BitmapAsset`, ensuring that the object is the correct size, and then draw all the data from an image. This captures all the data in the image as a bitmap and allows you to manipulate that data. In the following example, the `colorTransform` method manipulates the color data of the `BitmapData` object, and the

two bitmaps are merged via the `merge` method. The `colorTransform` method applies the data from a `ColorTransform` object to the `BitmapData` object. The `ColorTransform` object modifies the color of a display object or `BitmapData` according to the values passed in to the constructor:

```
ColorTransform(redMultiplier:Number = 1.0, greenMultiplier:Number = 1.0,
blueMultiplier:Number = 1.0, alphaMultiplier:Number = 1.0, redOffset:Number = 0,
greenOffset:Number = 0, blueOffset:Number = 0, alphaOffset:Number = 0)
```

When a `ColorTransform` object is applied to a display object, a new value for each color channel is calculated like this:

- New red value = (old red value * `redMultiplier`) + `redOffset`
- New green value = (old green value * `greenMultiplier`) + `greenOffset`
- New blue value = (old blue value * `blueMultiplier`) + `blueOffset`
- New alpha value = (old alpha value * `alphaMultiplier`) + `alphaOffset`

The `merge` method of the `BitmapData` class has the following signature:

```
merge(sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, redMultiplier
:uint, greenMultiplier:uint, blueMultiplier:uint, alphaMultiplier:uint):void
```

Its parameters are as follows:

`sourceBitmapData:BitmapData`
: The input bitmap image to use. The source image can be a different `BitmapData` object or the current `BitmapData` object.

`sourceRect:Rectangle`
: A rectangle that defines the area of the source image to use as input.

`destPoint:Point`
: The point within the destination image (the current `BitmapData` instance) that corresponds to the upper-left corner of the source rectangle.

`redMultiplier:uint`
: A hexadecimal `uint` value by which to multiply the red channel value.

`greenMultiplier:uint`
: A hexadecimal `uint` value by which to multiply the green channel value.

`blueMultiplier:uint`
: A hexadecimal `uint` value by which to multiply the blue channel value.

`alphaMultiplier:uint`
: A hexadecimal `uint` value by which to multiply the alpha transparency value.

A complete code listing follows with modifiable controls to alter the values of the `ColorTransform`:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="500" height="550"
creationComplete="imgMod()">
    <mx:Script>
        <![CDATA[
```

```
            import mx.core.BitmapAsset;
            import mx.controls.Image;


            [Embed(source="../assets/bigshakey.png")]
            private var shakey:Class;

            [Embed(source="../assets/mao.jpg")]
            private var mao:Class;

            //superimpose the two images together
            //using the vslider data
            private function imgMod():void
            {
                var maoData:BitmapData = new BitmapData(firstImg.width,
firstImg.height);
                var shakeyData:BitmapData = new BitmapData(secondImg.width,
secondImg.height);
                maoData.draw(firstImg);
                shakeyData.draw(secondImg);
                maoData.colorTransform(new Rectangle(0, 0, maoData.width,
maoData.height), new ColorTransform(redSlider.value/10, greenSlider.value/10,
blueSlider.value/10,alphaSlider.value/10));
                var red:uint = (uint(redSlider.value.toString(16)) / 10) * 160;
                var green:uint = (uint(greenSlider.value.toString(16)) / 10) * 160;
                var blue:uint = (uint(blueSlider.value.toString(16)) / 10) * 160;
                var alpha:uint = (uint(alphaSlider.value.toString(16)) / 10) * 160;
                shakeyData.merge(maoData, new Rectangle(0, 0, shakeyData.width,
shakeyData.height), new Point(0, 0), red, green, blue, alpha);
                mainImg.source = new BitmapAsset(shakeyData);
            }

        ]]>
    </mx:Script>
    <mx:HBox>
        <mx:Image id="firstImg" source="{mao}" height="200" width="200"/>
        <mx:Image id="secondImg" source="{shakey}" height="200" width="200"/>
    </mx:HBox>
    <mx:HBox>
        <mx:Text text="Red"/>
        <mx:VSlider height="100" id="redSlider" value="5.0" change="imgMod()"/>
        <mx:Text text="Blue"/>
        <mx:VSlider height="100" id="blueSlider" value="5.0" change="imgMod()"/>
        <mx:Text text="Green"/>
        <mx:VSlider height="100" id="greenSlider" value="5.0" change="imgMod()"/>
        <mx:Text text="Alpha"/>
        <mx:VSlider height="100" id="alphaSlider" value="5.0" change="imgMod()"/>
    </mx:HBox>
    <mx:Image id="mainImg"/>
</mx:VBox>
```

8.6 Apply a Convolution Filter to an Image

Problem

You want to allow users to alter the colors, contrast, or sharpness of an image.

Solution

Create an instance of a `ConvolutionFilter` and bind the properties of the matrix within the `ConvolutionFilter` to text inputs that the user can alter. Then push the filter onto the image's filters array to apply the filter.

Discussion

`ConvolutionFilter` is one of the most versatile and complex filters in the `flash.filters` package. It can be used to emboss, detect edges, sharpen, blur, and perform many other effects. All the parameters are controlled by a `Matrix` object representing a three-by-three matrix that is passed to the filter in its constructor. The `ConvolutionFilter` conceptually goes through each pixel in the source image one by one and determines the final color of that pixel by using the value of the pixel and its surrounding pixels. A matrix, specified as an array of numeric values, indicates to what degree the value of each particular neighboring pixel affects the final resulting value. The constructor is shown here:

```
ConvolutionFilter(matrixX:Number = 0, matrixY:Number = 0, matrix:Array = null,
divisor:Number = 1.0, bias:Number = 0.0, preserveAlpha:Boolean = true,
clamp:Boolean = true, co
lor:uint = 0, alpha:Number = 0.0)
```

Its parameters are as follows:

`matrixX:Number` *(default = 0)*
: The x dimension of the matrix (the number of columns in the matrix). The default value is 0.

`matrixY:Number` *(default = 0)*
: The y dimension of the matrix (the number of rows in the matrix). The default value is 0.

`matrix:Array` *(default = null)*
: The array of values used for matrix transformation. The number of items in the array must equal `matrixX` * `matrixY`.

`divisor:Number` *(default = 1.0)*
: The divisor used during matrix transformation. The default value is 1. A divisor that is the sum of all the matrix values evens out the overall color intensity of the result. A value of 0 is ignored and the default is used instead.

`bias:Number` *(default = 0.0)*
: The bias to add to the result of the matrix transformation. The default value is 0.

`preserveAlpha:Boolean` *(default = true)*
: A value of false indicates that the alpha value is not preserved and that the convolution applies to all channels, including the alpha channel. A value of true indicates that the convolution applies only to the color channels. The default value is true.

`clamp:Boolean` *(default = true)*
: For pixels that are off the source image, a value of true indicates that the input image is extended along each of its borders as necessary by duplicating the color values at the given edge of the input image. A value of false indicates another color should be used, as specified in the color and alpha properties. The default is true.

`color:uint` *(default = 0)*
: The hexadecimal color to substitute for pixels that are off the source image.

`alpha:Number` *(default = 0.0)*
: The alpha of the substitute color.

Some common effects for the `ConvolutionFilter` are as follows:

`new ConvolutionFilter(3,3,new Array(-5,0,1,1,-2,3,-1,2,1),1)`
: Creates an edge-detected image, where only areas of greatest contrast remain.

`new ConvolutionFilter(3,3,new Array(0,20,0,20,-80,20,0,20,0),10)`
: Creates a black-and-white outline.

`new ConvolutionFilter(5,5,new Array(0,1,2,1,0,1,2,4,2,1,2,4,8,4,2,1,2,4,2,1,0,1,2,1,0),50);`
: Creates a blur effect.

`new ConvolutionFilter(3,3,new Array(-2,-1,0,-1,1,1,0,1,2),0);`
: Creates an emboss effect.

The complete code listing is shown here:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="450" height="550">
    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;

            [Embed(source="../assets/mao.jpg")]
            private var mao:Class;

            private function convolve():void
            {
                var asset:BitmapAsset = new mao() as BitmapAsset;
                var convolution:ConvolutionFilter =
new ConvolutionFilter(matrixXSlider.value, matrixYSlider.value,
            [input1.text, input2.text, input3.text, input4.text, input5.text,
input6.text],
                divisorSlider.value, biasSlider.value, true);
                var _filters:Array = [convolution];
                asset.filters = _filters;
                img.source = asset;
            }
```

```
        ]]>
    </mx:Script>
    <mx:Button click="convolve()" label="convolve away"/>
    <mx:HBox>
        <mx:Text text="Matrix X"/>
        <mx:VSlider height="100" id="matrixXSlider" value="5.0" change="convolve()"/>
        <mx:Text text="Matrix Y"/>
        <mx:VSlider height="100" id="matrixYSlider" value="5.0" change="convolve()"/>
        <mx:Text text="Divisor"/>
        <mx:VSlider height="100" id="divisorSlider" value="5.0" change="convolve()"/>
        <mx:Text text="Bias"/>
        <mx:VSlider height="100" id="biasSlider" value="5.0" change="convolve()"/>
        <mx:VBox>
            <mx:TextInput id="input1" change="convolve()" width="40"/>
            <mx:TextInput id="input2" change="convolve()" width="40"/>
            <mx:TextInput id="input3" change="convolve()" width="40"/>
            <mx:TextInput id="input4" change="convolve()" width="40"/>
            <mx:TextInput id="input5" change="convolve()" width="40"/>
            <mx:TextInput id="input6" change="convolve()" width="40"/>
        </mx:VBox>
    </mx:HBox>
    <mx:Image id="img"/>
</mx:VBox>
```

8.7 Send Video to an FMS Instance via a Camera

Problem

You want to send a stream from the user's camera to a Flash Media Server (FMS) instance for use in a chat or other live media application.

Solution

Capture the user's camera stream by using the `flash.media.Camera.getCamera` method and then attach that camera to a `NetStream` that will be sent to the Flash Media Server instance. Use the `publish` method of the `NetStream` class to send the stream with a specified name to the application that will handle it.

Discussion

The `publish` method indicates to a Flash Media Server that has been connected to via the `NetConnection` class, that the `NetStream` will be sending information to the server. What the server does with that information depends on the application, but there are flags that can be set in the `publish` method that indicate to the server and the Flash Player what should be done with the streamed information. The `publish` method has the following signature:

```
publish(name:String = null, type:String = null):void
```

Its parameters are as follows:

`name:String` *(default = null)*
: A string that identifies the stream. If you pass false, the publish operation stops. Clients that subscribe to this stream must pass this same name when they call `NetStream.play`.

`type:String` *(default = null)*
: A string that specifies how to publish the stream. Valid values are `record`, `append`, and `live` (the default). If you pass `record`, Flash Player publishes and records live data, saving the recorded data to a new FLV file with a name matching the value passed to the `name` parameter. The file is stored on the server in a subdirectory within the directory that contains the server application. If the file already exists, it is overwritten. If you pass `append`, Flash Player publishes and records live data, appending the recorded data to an FLV file with a name that matches the value passed to the `name` parameter, stored on the server in a subdirectory within the directory that contains the server application. If no file with a name matching the `name` parameter is found, a file is created. If you omit this parameter or pass `live`, Flash Player publishes live data without recording it. If a file with a name that matches the value passed to the `name` parameter exists, the file is deleted.

When you record a stream by using Flash Media Server, the server creates an FLV file and stores it in a subdirectory in the application's directory on the server. Each stream is stored in a directory whose name matches the application instance name passed to `NetConnection.connect`. The server creates these directories automatically; you don't have to create one for each application instance. For example, the following code shows how you would connect to a specific instance of an application stored in a directory named lectureSeries in your application's directory. A file named lecture.flv is stored in a subdirectory named /yourAppsFolder/lectureSeries/streams/Monday:

```
var myNC:NetConnection = new NetConnection();
myNC.connect("rtmp://server.domain.com/lectureSeries/Monday");
var myNS:NetStream = new NetStream(myNC);
myNS.publish("lecture", "record");
```

If you don't pass a value for the instance name that matches, the value passed to the `name` property is stored in a subdirectory named /yourAppsFolder/appName/streams/_definst_ (for *default instance*).

This method can dispatch a `netStatus` event with several different information objects. For example, if someone is already publishing on a stream with the specified name, the `netStatus` event is dispatched with a code property of `NetStream.Publish.BadName`. For more information, see the `netStatus` event.

In the following example, the connection to the server is established, and the data from the camera is streamed to the server:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="500"
creationComplete="setUpCam()">
```

```
    <mx:Script>
        <![CDATA[

            private var cam:Camera;
            private var nc:NetConnection;
            private var ns:NetStream;

            private function setUpCam():void
            {
                trace(Camera.names.join(","));
                //I'm doing this only because it's the only way the
                //flash player will pick up the camera on my MacBook
                cam = flash.media.Camera.getCamera("2");
                vid.attachCamera(cam);
                nc = new NetConnection();
                nc.addEventListener(NetStatusEvent.NET_STATUS, netStatus);
                nc.connect("http://localhost:3002");
            }

            private function netStatus(event:NetStatusEvent):void
            {
                switch(event.info)
                {
                    case "NetConnection.Connect.Success":
                        ns = new NetStream(nc);
                        ns.attachCamera(cam, 20);
                        ns.attachAudio(Microphone.getMicrophone());
                        ns.publish("appname", "live");
                    break;
                }
            }

        ]]>
    </mx:Script>
    <mx:VideoDisplay id="vid" width="360" height="320"/>
</mx:Canvas>
```

8.8 Access a User's Microphone and Create a Sound Display

Problem

You want to access a user's microphone and use the sound level of the microphone to draw a sound level.

Solution

Access the microphone by using the `Microphone.getMicrophone` method. Access the sound level that this method detects by using the `mic.activityLevel` property of the `Microphone` class on a regular interval.

Discussion

The `Microphone` class provides access to a user's microphone and computer, and the user must allow the Flash Player application access for you to use the class. The `Microphone` class shows the level of sound that the microphone is detecting, and dispatches events when sound begins and when there has not been any sound for a given period of time.

Three properties of the `Microphone` class monitor and control the detection of activity. The read-only `activityLevel` property indicates the amount of sound the microphone is detecting on a scale from 0 to 100. The `silenceLevel` property specifies the amount of sound needed to activate the microphone and dispatch an `ActivityEvent.ACTIVITY` event. The `silenceLevel` property also uses a scale from 0 to 100, and the default value is 10. The `silenceTimeout` property describes the number of milliseconds that the activity level must stay below the silence level, until an `ActivityEvent.ACTIVITY` event is dispatched to indicate that the microphone is now silent. The default `silenceTimeout` value is 2000. Although both `Microphone.silenceLevel` and `Microphone.silenceTimeout` are read-only, you can change their values by using the `Microphone.setSilenceLevel` method.

The following example creates a `Microphone` object, which will prompt the user to accept or deny the Flash Player access to the microphone. Then, after microphone activity is detected via the `Activity` event, an enter frame event listener is added that will draw the `soundLevel` of the microphone into a `Canvas`.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="createMic()">

<mx:Script>
    <![CDATA[
    import flash.media.Microphone;
    import flash.events.ActivityEvent;
    import flash.events.Event;
    import flash.events.StatusEvent;

      public var mic:Microphone;

      public function createMic():void
      {
        mic = Microphone.getMicrophone();
        mic.setLoopBack(true);
        mic.addEventListener(ActivityEvent.ACTIVITY, activity);
        mic.addEventListener(StatusEvent.STATUS, status);
        mic.addEventListener(Event.ACTIVATE, active);
      }

      private function active(event:Event):void
      {
        trace(' active ');
      }
```

```
        private function status(event:StatusEvent):void
        {
          trace("status");
        }

        private function activity(event:ActivityEvent):void
        {
          trace("active ");
          addEventListener(Event.ENTER_FRAME, showMicLevel);
        }

        private function showMicLevel(event:Event):void
        {
          trace(mic.gain+" "+mic.activityLevel+" "+mic.silenceLevel+" "+mic.rate);
          level.graphics.clear();
          level.graphics.beginFill(0xccccff, 1);
          level.graphics.drawRect(0, 0, (mic.activityLevel * 30), 100);
          level.graphics.endFill();
        }

    ]]>
</mx:Script>
<mx:Canvas width="300" height="50" id="level"/>
</mx:VBox>
```

8.9 Smooth Video Displayed in a Flex Application

Problem

You need to control the smoothing of a video that is played back in an application.

Solution

Create a custom component that contains the `flash.media.Video` component, and then set `Video`'s `smoothing` property to true.

Discussion

To smooth video—that is, to make the video look less pixilated—you need to access the `flash.media.Video` object. Video smoothing, like image smoothing, requires more processing power than un-smoothed playback and can slow video playback for large or extremely high-quality videos.

The Flex `VideoDisplay` component does not allow you to set the `smoothing` property of the `flash.media.Video` object that it contains, so you must create a separate component that adds the lower-level Flash `Video` component and set the `smoothing` property:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300"
creationComplete="setup()">
    <mx:Script>
        <![CDATA[
```

```
            private var vid:Video;

            private var nc:NetConnection;
            private var ns:NetStream;
            private var metaDataObj:Object = {};

            private function setup():void {
                vid = new Video(this.width, this.height);
                vid.smoothing = true;
                this.rawChildren.addChild(vid);
                vid.y = 50;
                this.invalidateDisplayList();
            }

            private function startVid():void {
                nc = new NetConnection();
                nc.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
                nc.connect(null);
            }

            private function netStatusHandler(event:NetStatusEvent):void {
                ns = new NetStream(nc);
                metaDataObj.onMetaData = this.onMetaData;
                ns.client = metaDataObj;
                vid.attachNetStream(ns);
                ns.play("http://localhost:3001/Trailer.flv");
            }

            private function onMetaData(obj:Object):void {
                var i:int = 0;
                for each(var prop:Object in obj)
                {
                    trace(obj[i] + "  :  " + prop);
                    i++;
                }
                trace(obj.duration+" "+obj.framerate+" "+obj.bitrate);
            }

        ]]>
    </mx:Script>
    <mx:Button  click="startVid()" label="load" x="50"/>
    <mx:Button click="ns.resume()" label="resume" x="120"/>
    <mx:Button click="ns.pause()" label="pause" x="190"/>
</mx:Canvas>
```

8.10 Check Pixel-Level Collisions

Problem

You need to check whether images with alpha transparency regions are colliding with other images.

Solution

Draw the data of both images to a `BitmapData` object and use the `BitmapData.hitTest` method.

Discussion

The `BitmapData` object possesses a `hitTest` method that works similarly to the `hitTest` method defined by `DisplayObject` with one notable exception: whereas `DisplayObject`'s `hitTest` method returns true if the point given intersects with the bounds of the object, `BitmapData`'s `hitTest` method returns true if the pixel at the point given is above a certain threshold of alpha transparency. The signature of the method is shown here:

```
public function hitTest(firstPoint:Point, firstAlphaThreshold:uint, secondObject:
Object, secondBitmapDataPoint:Point = null, secondAlphaThreshold:uint = 1):Boolean
```

If an image is opaque, it is considered a fully opaque rectangle for this method. Both images must be transparent to perform pixel-level hit testing that considers transparency. When you are testing two transparent images, the alpha threshold parameters control what alpha channel values, from 0 to 255, are considered opaque. The method's parameters are as follows:

`firstPoint:Point`
: A position of the upper-left corner of the `BitmapData` image in an arbitrary coordinate space. The same coordinate space is used in defining the `secondBitmapPoint` parameter.

`firstAlphaThreshold:uint`
: The highest alpha channel value that is considered opaque for this hit test.

`secondObject:Object`
: A `Rectangle`, `Point`, `Bitmap`, or `BitmapData` object.

`secondBitmapDataPoint:Point` *(default = null)*
: A point that defines a pixel location in the second `BitmapData` object. Use this parameter only when the value of `secondObject` is a `BitmapData` object.

`secondAlphaThreshold:uint` *(default = 1)*
: The highest alpha channel value that's considered opaque in the second `BitmapData` object. Use this parameter only when the value of `secondObject` is a `BitmapData` object and both `BitmapData` objects are transparent.

In the following code sample, each corner of a rectangular image is checked for collisions against a PNG file with alpha transparency:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="1500" height="900">
    <mx:Script>
        <![CDATA[
            import flash.display.BlendMode;

            private var mainBmp:BitmapData;
```

```
            private var dragBmp:BitmapData;
            private var hasDrawn:Boolean = false;

            private function loaded():void{
                if(!hasDrawn){
                mainBmp = new BitmapData(mainImg.width, mainImg.height, true,
0x00000000);
                dragBmp = new BitmapData(dragImg.width, dragImg.height, true,
0x00000000);
                hasDrawn = true;
                this.addEventListener(Event.ENTER_FRAME, showHits);
                }
            }

            private function showHits(event:Event):void
            {
                mainBmp.draw(mainImg);
                dragBmp.draw(dragImg);
                if(mainBmp.hitTest(new Point(0,0), 0xff, dragImg.getBounds(this).
topLeft)){
                    trace(" true ");
                    return;
                }
                if(mainBmp.hitTest(new Point(0,0), 0xff, dragImg.getBounds(this).
bottomRight)){
                    trace(" true ");
                    return;
                }
                if(mainBmp.hitTest(new Point(0,0), 0xff, new Point(dragImg.getBounds
(this).left, dragImg.getBounds(this).bottom))){
                    trace(" true ");
                    return;
                }
                if(mainBmp.hitTest(new Point(0,0), 0xff, new Point(dragImg.getBounds
(this).right, dragImg.getBounds(this).top))){
                    trace(" true ");
                    return;
                }
                trace(" false ");
            }

        ]]>
    </mx:Script>
    <mx:Image id="mainImg" source="../assets/alphapng.png" cacheAsBitmap="true"/>
    <mx:Image cacheAsBitmap="true" id="dragImg" mouseDown="dragImg.startDrag(false,
this.getBounds(stage)), loaded()" rollOut="dragImg.stopDrag()"
mouseUp="dragImg.stopDrag()" source="../assets/bigshakey.png"/>

</mx:Canvas>
```

This code returns false when the pixels of the first image at the given points do not possess alpha values greater than those set in the `hitTest` method. In Figure 8-1, the two light blue squares are within a PNG file with alpha transparency. The shake is a separate image that, at this moment, is not colliding with an area of the PNG with a

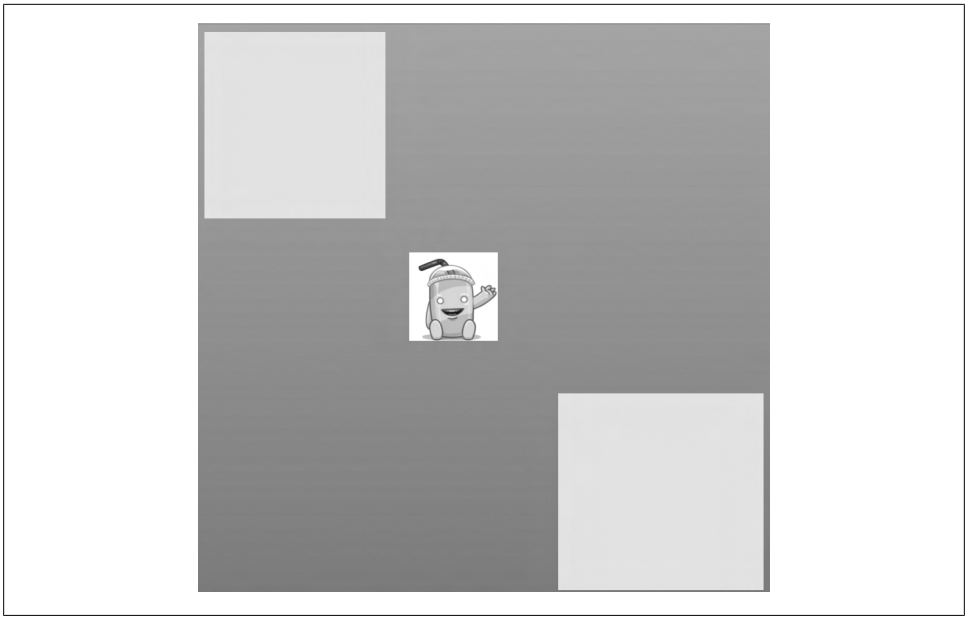

Figure 8-1. hitTest() will return false.

high-enough alpha. In Figure 8-2, however, the shake collides with a square and the method returns true.

8.11 Read and Save a User's Webcam Image

Problem

You want to read an image from a user's webcam and save that image to a server.

Solution

Create a `Camera` object and attach it to a `Video` object. Then create a button that will read a bitmap from the `Video` object and save the bitmap data to a server-side script that will save the image.

Discussion

To capture an image from a webcam, create a bitmap from the `Video` object that is displaying the camera image. The Flash Player doesn't provide any access to the stream of data that is read from the webcam, however, so you need to render the data as a bitmap before you can use it.

After the image has been captured as a `BitmapData` object, you can pass that data to an instance of the `JPEGEncoder` class to convert the image into JPEG data. Next, save the

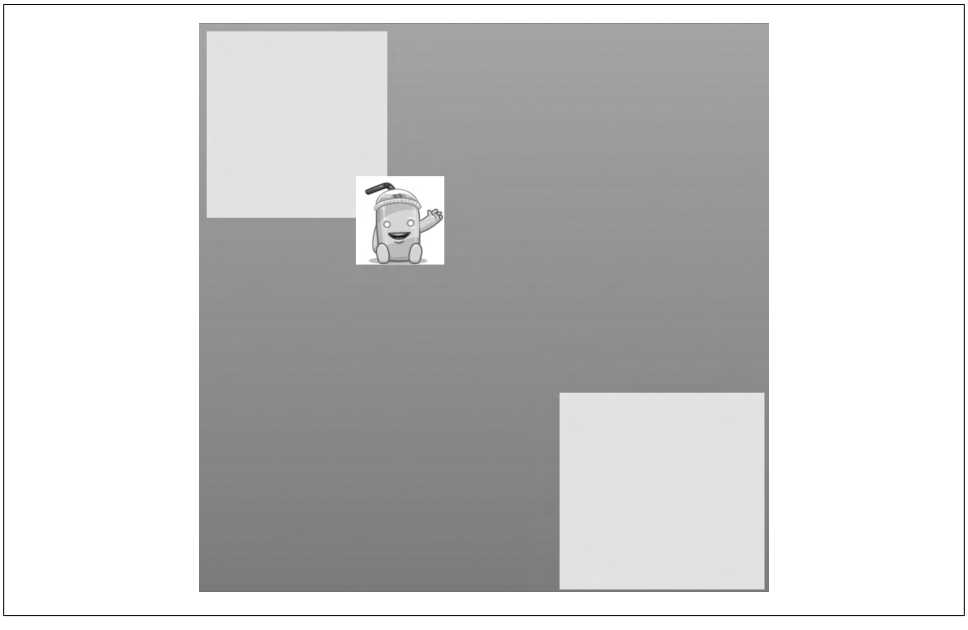

Figure 8-2. hitTest() will return true.

JPEG to a server by adding the data to a `URLRequest` object and sending it via the `navigateToURL` method. For example:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="500"
creationComplete="setUpCam()">
    <mx:Script>
        <![CDATA[
            import flash.net.navigateToURL;
            import flash.net.sendToURL;

            import mx.graphics.codec.JPEGEncoder;

            private var cam:Camera;

            private function setUpCam():void {
                cam = flash.media.Camera.getCamera("2");
                vid.attachCamera(cam);
            }

            private function saveImage():void {
                var bitmapData:BitmapData = new BitmapData(vid.width, vid.height);
                bitmapData.draw(vid);
                var encode:JPEGEncoder = new JPEGEncoder(100);
                var ba:ByteArray = encode.encode(bitmapData);
                var urlRequest:URLRequest = new URLRequest("/jpg_reader.php");
                urlRequest.method = "POST";
                var urlVars:URLVariables = new URLVariables();
                urlVars.pic = ba;
```

```
                urlRequest.data = urlVars;
                flash.net.navigateToURL(urlRequest, "_blank");
            }

        ]]>
    </mx:Script>
    <mx:VideoDisplay id="vid" width="360" height="320"/>
    <mx:Button label="Take Picture Now" click="saveImage()"/>
</mx:Canvas>
```

8.12 Use Blend Modes with Multiple Images

Problem

You want to blend multiple images.

Solution

Set the `blendMode` property of the images.

Discussion

Every `DisplayObject` defines a `blendMode` property that controls how that `DisplayObject` appears, controlling the alpha and how any `DisplayObject`s beneath that object in the `DisplayList` appear through that component. The blend modes should be familiar to anyone who has worked with Adobe Photoshop or After Effects:

`BlendMode.ADD ("add")`
: Creates an animated lightening dissolve effect between two images.

`BlendMode.ALPHA ("alpha")`
: Applies the transparency of the foreground to the background.

`BlendMode.DARKEN ("darken")`
: Superimposes type.

`BlendMode.DIFFERENCE ("difference")`
: Creates more-vibrant colors.

`BlendMode.ERASE ("erase")`
: Erases part of the background by using the foreground alpha.

`BlendMode.HARDLIGHT ("hardlight")`
: Creates shading effects.

`BlendMode.INVERT ("invert")`
: Inverts the background.

`BlendMode.LAYER ("layer")`
: Forces the creation of a temporary buffer for precomposition for a particular display object.

`BlendMode.LIGHTEN ("lighten")`
: Superimposes type.

`BlendMode.MULTIPLY ("multiply")`
: Creates shadows and depth effects.

`BlendMode.NORMAL ("normal")`
: Specifies that the pixel values of the blend image override those of the base image.

`BlendMode.OVERLAY ("overlay")`
: Creates shading effects.

`BlendMode.SCREEN ("screen")`
: Creates highlights and lens flares.

`BlendMode.SUBTRACT ("subtract")`
: Creates an animated darkening dissolve effect between two images.

The following example applies the various blend modes to the two `Image` objects:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="800" height="800">
    <mx:Script>
        <![CDATA[
            import flash.display.BlendMode;
        ]]>
    </mx:Script>
    <mx:Image id="img1" mouseDown="img1.startDrag(false, this.getBounds(stage)),
swapChildren(img1, img2)" rollOut="img1.stopDrag()" mouseUp="img2.stopDrag()"
source="../assets/mao.jpg"/>
    <mx:Image id="img2" mouseDown="img2.startDrag(false, this.getBounds(stage)),
swapChildren(img2, img1)" rollOut="img2.stopDrag()" mouseUp="img2.stopDrag()"
source="../assets/bigshakey.png"/>
    <mx:HBox>
        <mx:CheckBox id="chb" label="which one"/>
        <mx:ComboBox id="cb" dataProvider="{[BlendMode.ADD, BlendMode.ALPHA, BlendMode
.DARKEN, BlendMode.DIFFERENCE, BlendMode.ERASE, BlendMode.HARDLIGHT, BlendMode.
INVERT,BlendMode.LAYER, BlendMode.LIGHTEN, BlendMode.MULTIPLY, BlendMode.NORMAL,
BlendMode.OVERLAY, BlendMode.SCREEN, BlendMode.SUBTRACT]}"
         change="chb.selected ? img1.blendMode = cb.selectedItem as String :
img2.blendMode = cb.selectedItem as String"/>
    </mx:HBox>
</mx:Canvas>
```

8.13 Handle Cue Points in FLV Data

Problem

You need to work with cue points that are embedded in an FLV file while it plays.

Solution

Use the `onCuePoint` event of the `NetStream` class to create a handler method to be fired whenever a cue point is encountered.

Discussion

A *cue point* is a value inserted into an FLV file at a certain time within a video that contains either simply a name or a data object with a hash table of values. Usually cue points are inserted into an FLV when the file is being encoded, and any values are determined there. The Flex `VideoDisplay` object uses the `mx.controls.videoclasses.CuePoint` manager class to handle detecting and reading any data from a cue point. For a more-complete understanding of this, consider an example using the `flash.media.Video` object.

When the `NetConnection` object has connected and the `NetStream` is being instantiated, you need to set an object to relay any metadata and cue point events to handler methods:

```
var obj:Object = new Object();
obj.onCuePoint = onCuePoint;
obj.onMetaData = onMetaData;
ns.client = obj;
```

This needs to occur before the `NetStream play` method is called. Note in the following code that both the `onMetaData` and `onCuePoint` events accept an object as a parameter:

```
import flash.events.NetStatusEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;
import mx.core.UIComponent;

public class CuePointExample extends UIComponent
{
    private var ns:NetStream;
    private var nc:NetConnection;
    private var obj:Object = {};
    private var vid:Video;

    public function CuePointExample () {
        super();
        vid = new Video();
        addChild(vid);
        nc = new NetConnection();
        nc.addEventListener(NetStatusEvent.NET_STATUS, netStatusEventHandler);
        nc.connect(null);
    }

    private function netStatusEventHandler(event:NetStatusEvent):void {
        ns = new NetStream(nc);
        obj.onCuePoint = onCuePoint;
        obj.onMetaData = onMetaData;
        ns.client = obj;
        ns.play("http://localhost:3001/test2.flv");
        vid.attachNetStream(ns);
    }

    private function onCuePoint(obj:Object):void {
        trace(obj.name+" "+obj.time+" "+obj.length+" ");
```

```
            for each(var o:String in obj.parameters) {
                trace(obj[o]+" "+o);
            }
        }

        private function onMetaData(obj:Object):void{
        }
    }
```

Using the `mx.controls.VideoDisplay` simplifies working with a cue point object quite substantially. When using the `CuePointEvent` dispatched by the `CuePointManager`, unlike in the preceding case, the received event possesses only three properties: `cuePoint Time`, `cuePointName`, and `cuePointType`. If you need more or different information from the cue point, you can write a custom class to return the cue point data and set it to the `cuePointManager` property of the `VideoDisplay` object. The complete code listing is shown here:

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    <mx:Script>
        <![CDATA[
            import mx.events.CuePointEvent;

            private function onCuePoint(event:CuePointEvent):void {
                trace(event.cuePointName+" "+event.cuePointTime+"
"+event.cuePointType+" ");
            }

        ]]>
    </mx:Script>
    <mx:VideoDisplay id="vid" cuePoint="onCuePoint(event)"/>
</mx:VBox>
```

8.14 Create a Video Scrubber

Problem

You need to create a control that a user can use to scrub through a video as it plays.

Solution

Create a draggable `Sprite` object and listen for any `DragEvent` events dispatched from it. In the event handler for the `DragEvent`, set the amount to seek forward or backward in the `NetStream` that is streaming the video to the `Video` object.

Discussion

You can use any draggable display object to set the new position at which the video should be played. In this example, the `seek` method of the `NetStream` begins playback from the specified point in seconds from the beginning of the video:

```
ns.seek((playhead.x/timeline.width) * length);
```

To determine which second in the video that the user meant to seek, divide the position of the dragged `Sprite` by the width of the timeline area and multiply by the length of the video. The `NetStream` will take care of locating the appropriate frames in the video and restarting the streaming from that point.

```
import flash.display.Sprite;
import flash.events.MouseEvent;
import flash.events.NetStatusEvent;
import flash.media.Video;
import flash.net.NetConnection;
import flash.net.NetStream;

import mx.core.UIComponent;

public class Scrubber extends UIComponent
{

    private var playhead:Sprite;
    private var timeline:Sprite;
    private var ns:NetStream;
    private var nc:NetConnection;
    private var obj:Object = {};
    private var length:int;
    private var vid:Video;

    public function Scrubber () {
        super();
        playhead = new Sprite();
        addChild(playhead);
        playhead.graphics.beginFill(0x0000ff, 1);
        playhead.graphics.drawCircle(0, 0, 5);
        playhead.graphics.endFill();
        playhead.addEventListener(MouseEvent.MOUSE_DOWN, startSeek);
        timeline = new Sprite();
        timeline.graphics.beginFill(0xcccccc, 1);
        timeline.graphics.drawRect(0, 0, 200, 10);
        timeline.graphics.endFill();
        addChild(timeline);
        timeline.addChild(playhead);
        playhead.y = 4;
        vid = new Video();
        addChild(vid);
        vid.y = 100;

        nc = new NetConnection();
        nc.addEventListener(NetStatusEvent.NET_STATUS, netStatus);
        nc.connect(null);
    }

    private function netStatus(event:NetStatusEvent):void {
        obj.onMetaData = onMetaData;
        ns = new NetStream(nc);
        ns.client = obj;
        vid.attachNetStream(ns);
        ns.play("http://localhost:3001/test.flv");
```

```
    }

    private function onMetaData(obj:Object):void {
        length = obj.duration;
        trace(length);
    }

    private function startSeek(mouseEvent:MouseEvent):void {
        playhead.startDrag(false, timeline.getBounds(this));
        addEventListener(MouseEvent.MOUSE_MOVE, seek);
        playhead.addEventListener(MouseEvent.ROLL_OUT, endSeek);
        playhead.addEventListener(MouseEvent.MOUSE_UP, endSeek);
    }

    private function seek(mouseEvent:MouseEvent):void {
        ns.seek((playhead.x/timeline.width) * length);
    }

    private function endSeek(mouseEvent:MouseEvent):void {
        removeEventListener(MouseEvent.MOUSE_MOVE, seek);
        playhead.stopDrag();
    }
```

8.15 Read ID3 Data from an MP3 File

Problem

You want to read ID3 data from an MP3 file.

Solution

Use the `Event.ID3` method that the `Sound` class will dispatch when the ID3 data has been parsed.

Discussion

The `Sound` class dispatches an event when the ID3 data has been parsed from a loaded MP3 file. That data is then stored as an `ID3Info` object, which defines variables to access all the properties written into the initial bytes of the MP3:

```
private var sound:Sound;

public function _8_16()
{
    sound = new Sound();
    sound.addEventListener(Event.ID3, onID3InfoReceived);
    sound.load(new URLRequest("../assets/1.mp3"));
}

private function onID3InfoReceived(event:Event):void
{
    var id3:ID3Info = event.target.id3;
```

```
        for (var propName:String in id3)
        {
            trace(propName + " = " + id3[propName]);
        }
    }
```

The information from a song I was listening to while I wrote this recipe appears like this:

```
TCON = Alternative & Punk
TIT2 = The Pink Batman
TRCK = 2/9
TPE1 = Dan Deacon
TALB = Spiderman Of The Rings
TCOM = Dan Deacon
```

The ID3 info of an MP3 file is simply a grouping of bytes in a certain order that are read and turned into strings or integers. MP3 is the only file format that the Flash Player supports out of the box. Developer Benjamin Dobler of RichApps (www.richapps.de), however, has done some exceptional work with the WAV format. Getting the WAV file to play back in the Flash Player is slightly more tricky. If you're interested, go to Benjamin's site and take a look. If you want to parse the data from a WAV file, it looks like this:

```
public var bytes:ByteArray;
public var chunkId:String;
public var chunkSize:int;
public var chunkFormat:String;
public var subchunk1Id:String;
public var subchunk1Size;
public var audioFormat;
public var channels;
public var sampleRate;
public var bytesPersecond;
public var blockAlign;
public var bitsPerSample;
public var dataChunkSignature:String;
public var dataChunkLength;

public function read(bytes:ByteArray):void{
    this.bytes = bytes;
    // Read Header
    bytes.endian = "littleEndian";
    chunkId = bytes.readMultiByte(4,"utf"); //RIFF
    chunkSize = bytes.readUnsignedInt();
    chunkFormat = bytes.readMultiByte(4,"utf"); //WAVE
    subchunk1Id = bytes.readMultiByte(4,"iso-8859-1"); // 12 Header Signature
    subchunk1Size = bytes.readInt(); // 16 4 <fmt length>
    audioFormat = bytes.readShort(); // 20 2 <format tag> sample
    channels = bytes.readShort(); // 22     2 <channels> 1 = mono, 2 = stereo
    sampleRate = bytes.readUnsignedInt();// 24     4 <sample rate>
    bytesPersecond = bytes.readUnsignedInt(); //28 4 <bytes/second>     Sample-Rate *
Block-Align
    blockAlign = bytes.readShort(); // 32 2 <block align> channel * bits/sample / 8
    bitsPerSample = bytes.readUnsignedShort(); //34 2 <bits/sample> 8, 16 or 24
    dataChunkSignature = bytes.readMultiByte(4,"iso-8859-1"); //RIFF
```

```
        dataChunkLength = bytes.readInt();
    }
```

If you want to read the header info from an AU file, it would look like this:

```
    public var bytes:ByteArray;
    public var magicId;
    public var header;
    public var datasize;
    public var channels;
    public var comment;
    public var sampleRate;
    public var encodingInfo;

    public function read(bytes:ByteArray):void{
        this.bytes = bytes;
        // Read Header
        bytes.endian = "bigEndian";
        magicId = bytes.readUnsignedInt();
        header = bytes.readInt();
        datasize = bytes.readUnsignedInt();
        encodingInfo = bytes.readUnsignedInt();
        sampleRate = bytes.readUnsignedInt();
        channels = bytes.readInt();
        comment = bytes.readMultiByte(uint(header)-24, "utf");
    }
```

MP3 files may be the easiest format from which to read data, but they are certainly not the only format from which you can read.

8.16 Display a Custom Loader while Loading Images

Problem

You want to display custom animation while an image loads.

Solution

Create a custom graphic and listen for the `ProgressEvent.PROGRESS` event from the `Image` object loading the image. Then draw into the graphic by using the `bytesLoaded` and `bytesTotal` properties.

Discussion

There are two approaches to displaying an image when using the `Image` component: You can set the source for the `Image` class in MXML or you can pass a URL to load and use the `img.load` method:

```
    img.load("http://thefactoryfactory.com/beach.jpg");
```

Before you load the image, though, you want to attach an event listener to ensure that each `ProgressEvent` is handled:

```
img.addEventListener(ProgressEvent.PROGRESS, progress);
```

In the `progress` method, which is handling the `ProgressEvent.PROGRESS` event, a `UIComponent` is redrawn by using the `bytesLoaded` property of the `Image`:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" creationComplete="loadImage()">
    <mx:Script>
        <![CDATA[

            private var m:Matrix;

            private function loadImage():void {
                var m:Matrix = new Matrix();
                m.createGradientBox(450, 40);
                img.addEventListener(ProgressEvent.PROGRESS, progress);
                img.load("http://thefactoryfactory.com/beach.jpg");
            }

            private function progress(event:Event):void{
                grid.graphics.clear();
                grid.graphics.beginGradientFill("linear", [0x0000ff, 0xffffff],
[1, 1], [0x00, 0xff], m);
                grid.graphics.drawRect(0, 0, (img.bytesLoaded / img.bytesTotal) * 300,
40);
                grid.graphics.endFill();
            }

        ]]>
    </mx:Script>
    <mx:Canvas id="grid" height="40" width="300"/>
    <mx:Image id="img" y="40"/>
</mx:Canvas>
```

8.17 Enable Image Upload in Flex

Problem

You want to enable users to upload images via Flex to be stored on a server.

Solution

Create a `FileReference` object and attach the appropriate filters so that users can upload the correct image types only. Then listen for the complete handler from the `FileReference` object and send the uploaded image to a server-side script.

Discussion

Image upload in Flex as well as in Flash relies on the use of the `FileReference` class. The `FileReference` object, when invoked, creates a window by using the browser's normal upload window and graphic and sends the image through the Flash Player when

the user has selected a file for upload. Add an event listener to the `FileReference` object to indicate that the user has selected a file:

```
fileRef.addEventListener(Event.SELECT, selectHandler);
```

Then add a method to upload the file that the user has selected:

```
private function selectHandler(event:Event):void {
            var request:URLRequest = new URLRequest("http://thefactoryfactory.com/
upload2.php");
            fileRef.upload(request, "Filedata", true);
        }
```

After the file has been uploaded, send it to a PHP script to save the uploaded image:

```
package oreilly.cookbook
{
    import mx.core.UIComponent;
    import flash.net.FileFilter;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.events.Event;

    public class _8_17 extends UIComponent
    {

        private var fileRef:FileReference;

        public function _8_17() {
            super();
            startUpload();
        }

        private function startUpload():void {
            //set all the file types we're going to allow the user to upload
            var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif,
*.png)", "*.jpg; *.jpeg; *.gif; *.png");
            var allTypes:Array = new Array(imageTypes);
            fileRef = new FileReference();
            fileRef.addEventListener(Event.SELECT, selectHandler);
            fileRef.addEventListener(Event.COMPLETE, completeHandler);
            //tell the FileRefence object to accept only those image
            //types
            fileRef.browse(allTypes);
        }

        private function selectHandler(event:Event):void {
            var request:URLRequest = new URLRequest("http://thefactoryfactory.com/
upload2.php");
            fileRef.upload(request, "Filedata", true);
        }
        private function completeHandler(event:Event):void {
            trace("uploaded");
        }
    }
}
```

Because the file has already been uploaded, you can deal with the data on the server, moving the file to (in this case) a folder called images:

```
$file_temp = $_FILES['file']['tmp_name'];
$file_name = $_FILES['file']['name'];
$file_path = $_SERVER['DOCUMENT_ROOT']."/images";
//checks for duplicate files
if(!file_exists($file_path."/".$file_name)) {
    //complete upload
    $filestatus = move_uploaded_file($file_temp,$file_path."/".$file_name);
    if(!$filestatus) {
        //error in uploading file
    }
}
```

8.18 Compare Two Bitmap Images

Problem

You need to compare two bitmap images and display the differences between them.

Solution

Read the bitmap data from two images and use the `compare` method to compare the two images. Set the difference of the two images as the source of a third image.

Discussion

The `compare` method of the `BitmapData` class returns a `BitmapData` object that contains all the pixels that do not match in two specified images. If the two `BitmapData` objects have the same dimensions (width and height), the method returns a new `BitmapData` object, in which each pixel is the difference between the pixels in the two source objects: If two pixels are equal, the difference pixel is 0x00000000. If two pixels have different RGB values (ignoring the alpha value), the difference pixel is 0xFFRRGGBB, where RR/GG/BB are the individual difference values between red, green, and blue channels. Alpha channel differences are ignored in this case. If only the alpha channel value is different, the pixel value is 0xZZFFFFFF, where ZZ is the difference in the alpha value.

```
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="800">
    <mx:Script>
        <![CDATA[
            import mx.core.BitmapAsset;

            private function compare():void {
                var bmpd1:BitmapData = new BitmapData(img1.width, img1.height);
                var bmpd2:BitmapData = new BitmapData(img2.width, img2.height);
                bmpd1.draw(img1)
                bmpd2.draw(img2);
                var diff:BitmapData = bmpd2.compare(bmpd1) as BitmapData;
                var bitmapAsset:BitmapAsset = new BitmapAsset(diff);
```

```
                img3.source = bitmapAsset;
            }

        ]]>
    </mx:Script>
    <mx:Image id="img1" source="../assets/mao.jpg" height="200" width="200"/>
    <mx:Image id="img2" source="../assets/bigshakey.png" height="200" width="200"/>
    <mx:Button click="compare()" label="compare"/>
    <mx:Image id="img3"/>
</mx:VBox>
```